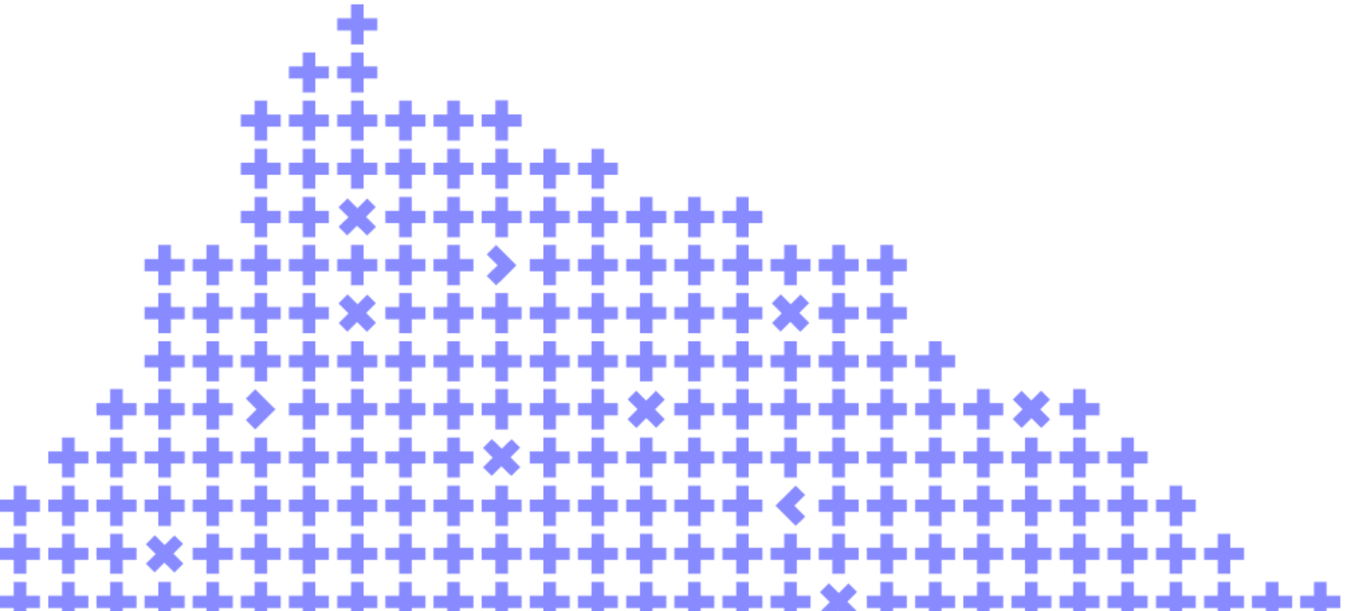
Practical aspects of B+ trees

Nikolay Izhikov



++ High Load Armenia

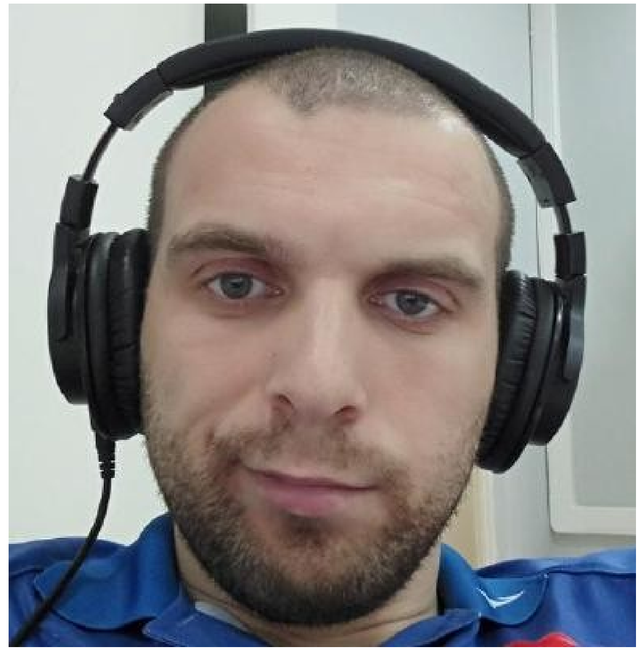
Co-organizer

Yandex

Nikolay Izhikov

I write code

- Apache Ignite PMC
- Apache Kafka contributor
- https://t.me/db_links
- https://t.me/nizhikovTalks
- https://github.com/nizhikov
- nizhikov@apache.org



Speech structure

- BTree (B+tree, B*tree, Blink tree, etc.) design principles.
- Basic tree operation implementation.
- Concurrent tree design.

B-trees design principles

There are two core design limitations:

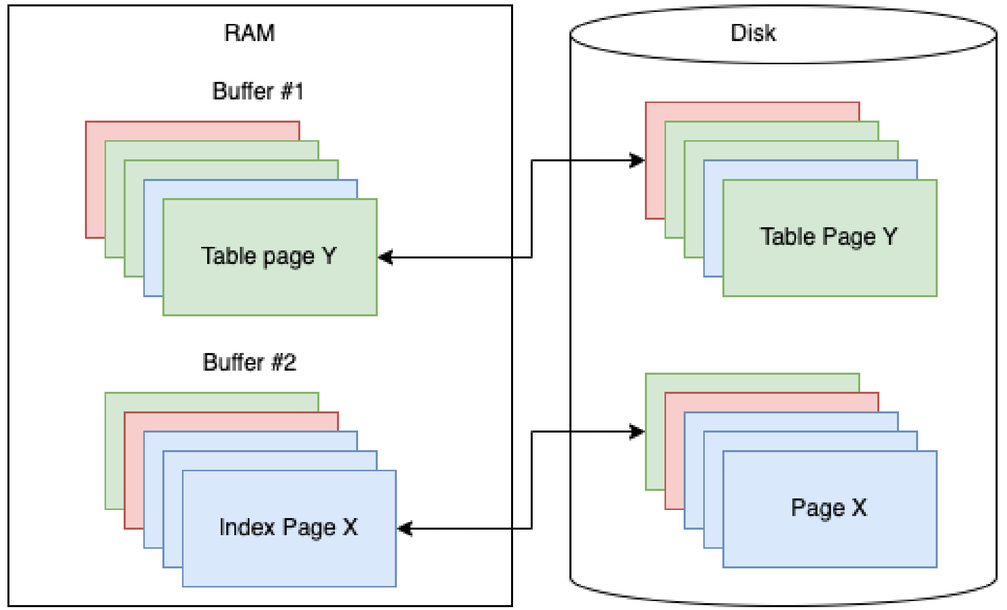
- Hardware.
- Security Officer.

Database features:

- Store on the disk unless you in-memory DB.
- Search:
  - Lookup.
  - Bounded iteration BETWEEN x AND Y clause.

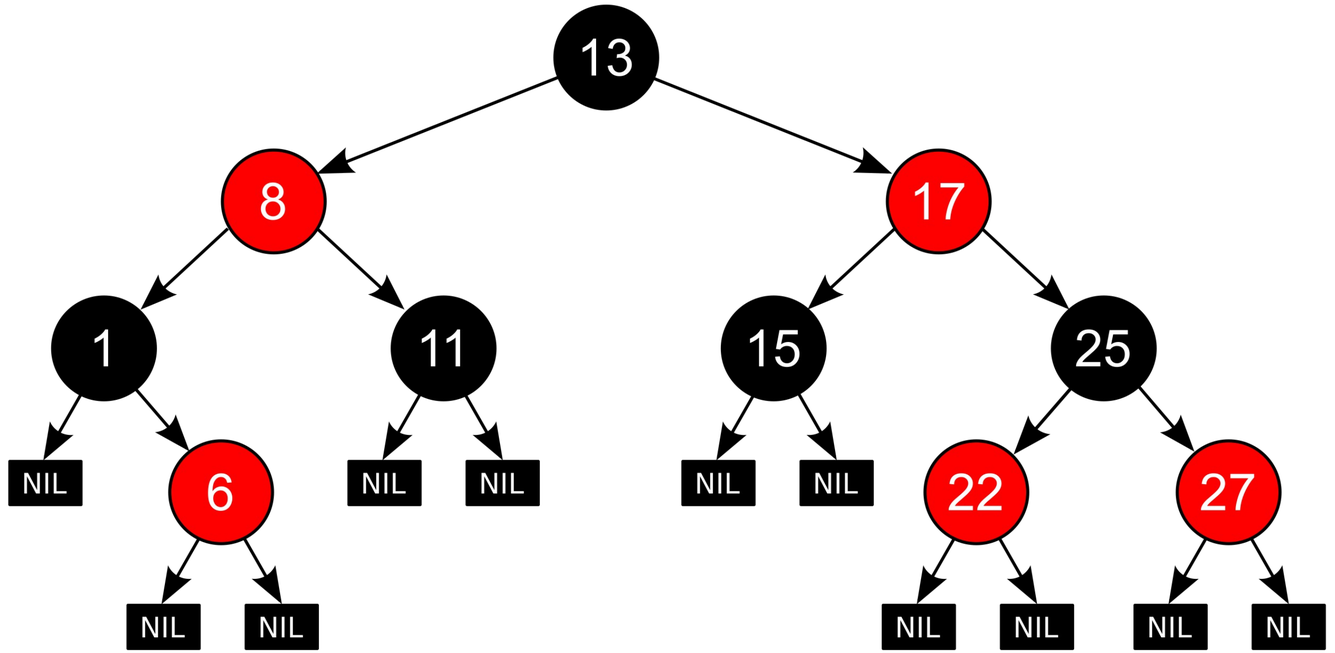
Database storage basics

- Storage unit is a page.
- Page size aligned with the disk storage unit.
- Pages combined in reusable buffers.
- Different page types:
  - Index pages
  - Table pages
  - Metadata pages
- WAL (write ahead log) stores records delta.
- Pages written as a whole periodically during checkpoint or similar process.



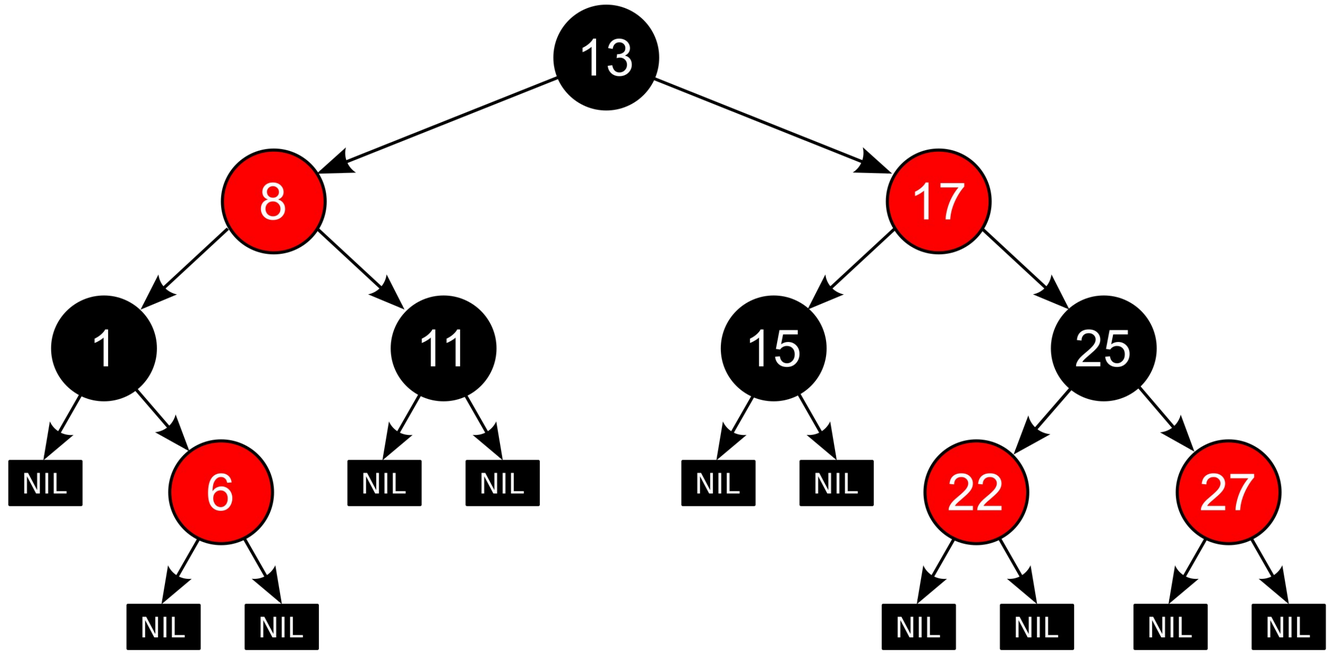
Binary tree (AVL or Red-Black)

- Good concurrency.
- Easy to implement.
- Good performance:
  - Search - O (log n).
  - Insert - O (log n).
  - Remove - O (log n).



Binary tree (AVL or Red-Black)

- Bad hardware mapping.



Tree properties

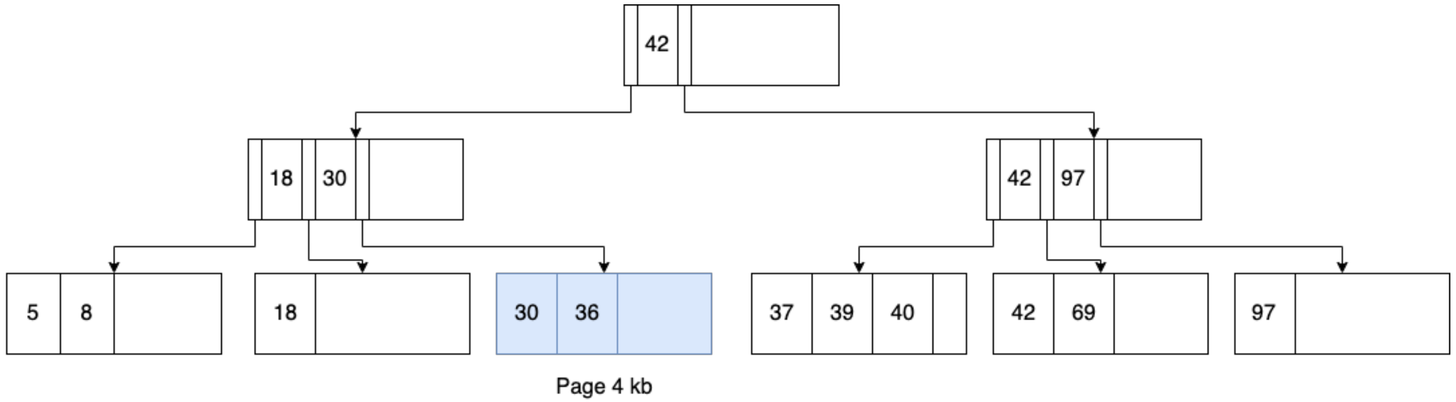
- All leaves at the same level.
- The root has at least two children.
- Each node except root can have a maximum of m children and at least n children.
- Each node can contain a maximum of m-1 keys and a minimum of n - 1 keys.

Advantages

- Small tree height on practice.
- Good performance:
  - All leafs on the same level - same search performance for each key.
  - Insertion, remove rarely modifies more then one page.
- Hardware friendly structure.

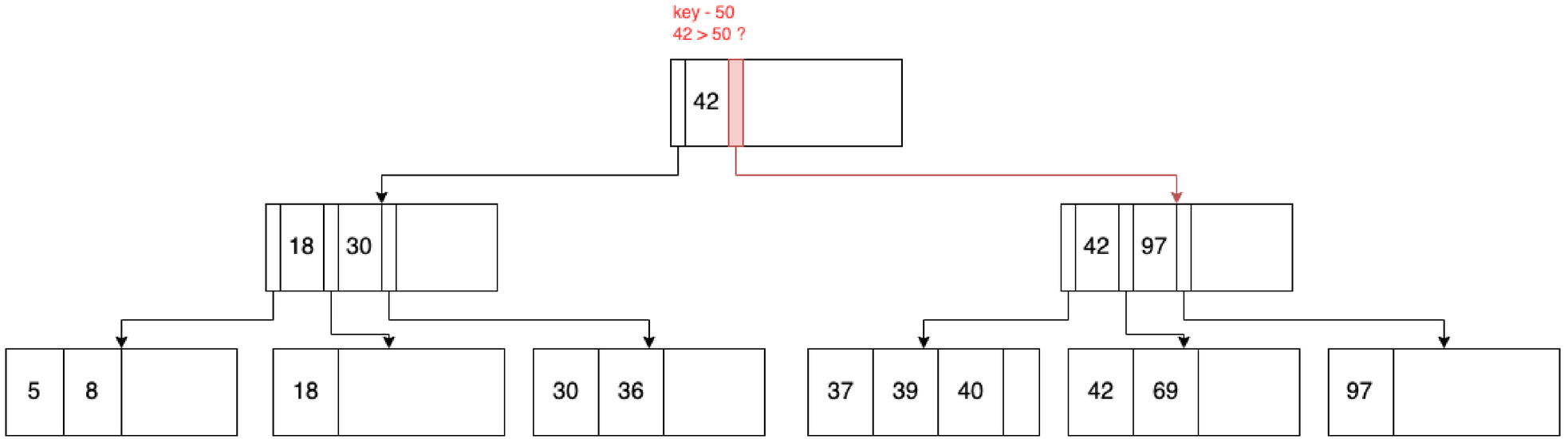
Example

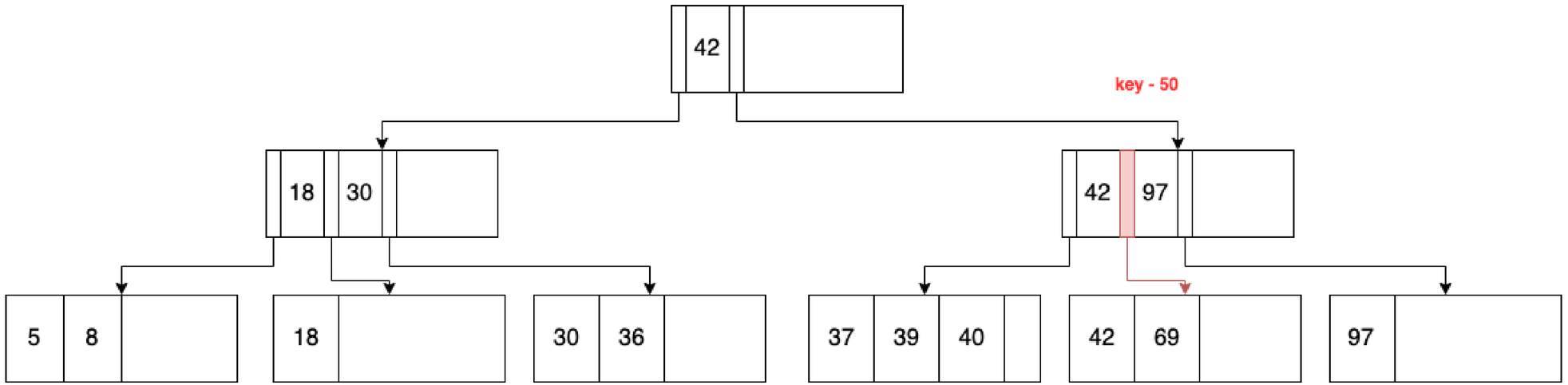
- Each node size aligned with the disk write unit - 2kb, 4kb, etc.
- Several keys stored in each node.
- Inner node store tuple (key, page-link).
- Only leaf node stores actual data - (key, row-link).
- Keys sorted inside node.

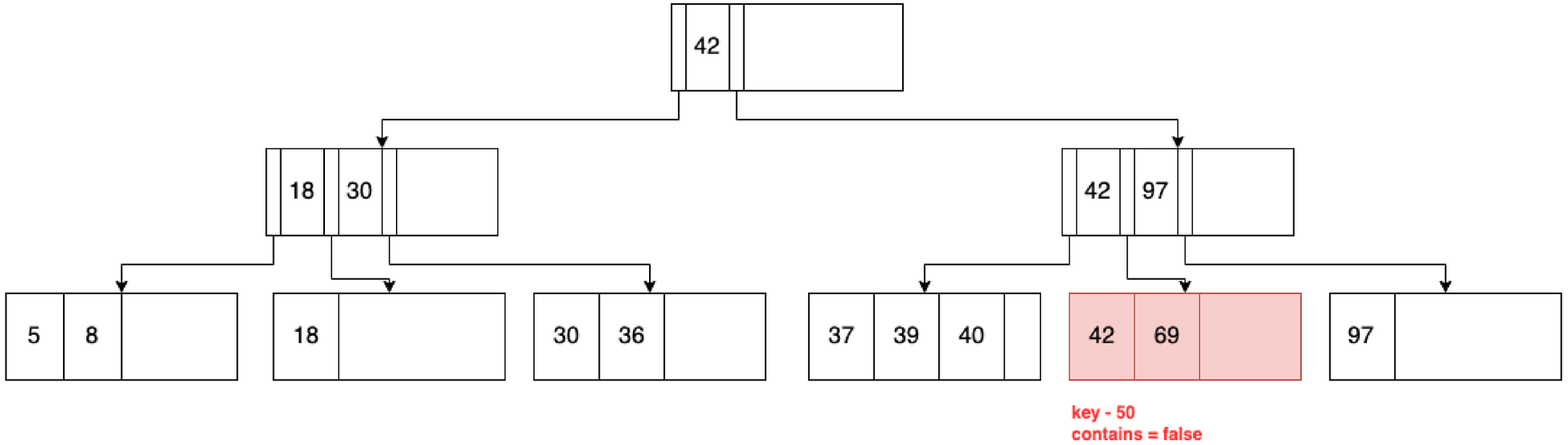


Tree operations

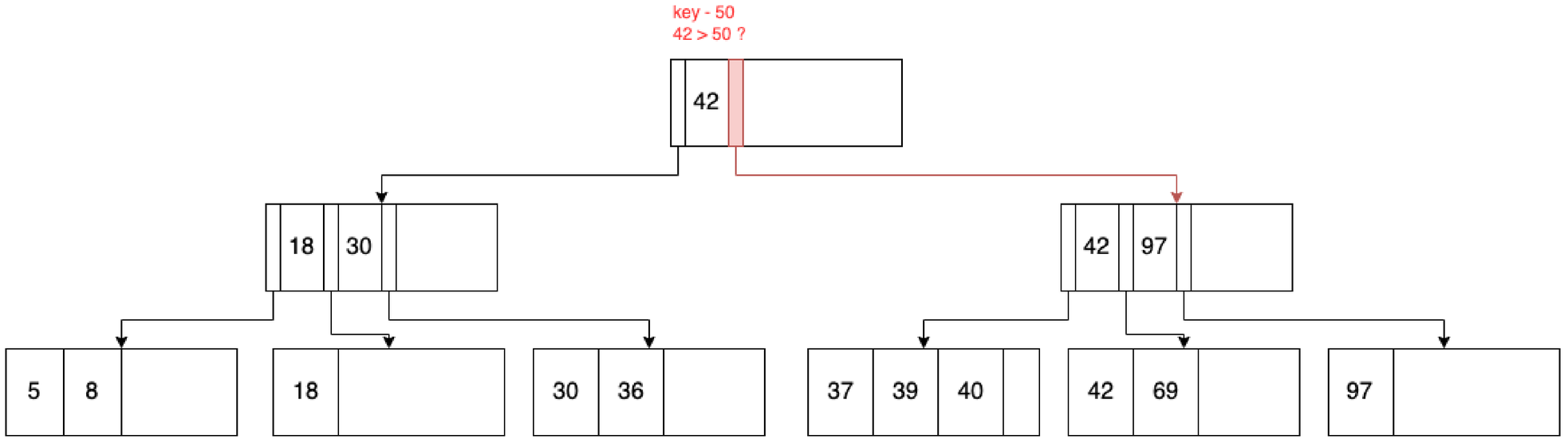
Search (contains)

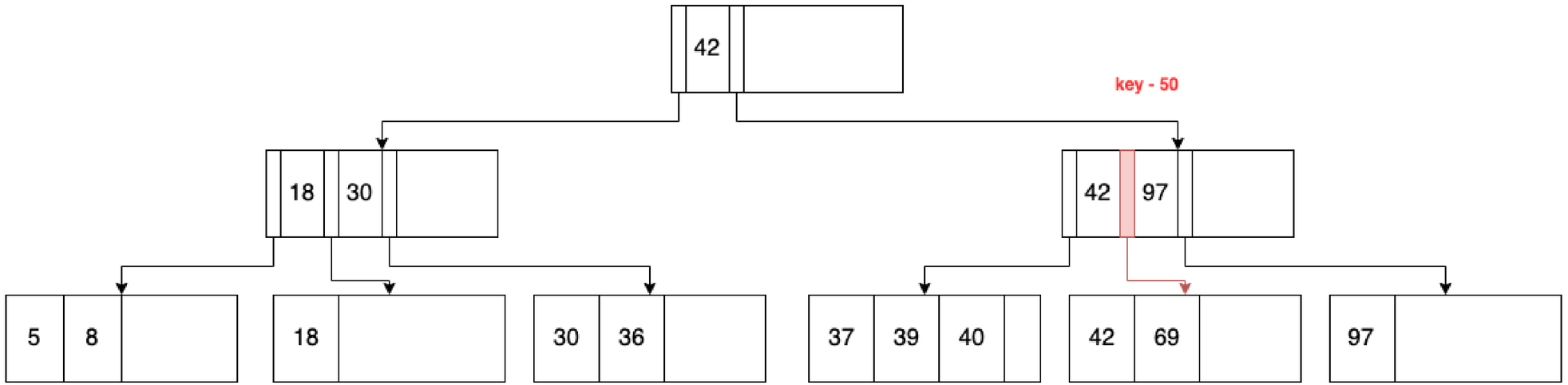

key - 50
42 > 50 ?
42
18
30
42
97
5
8
18
30
36
37
39
40
42
69
97

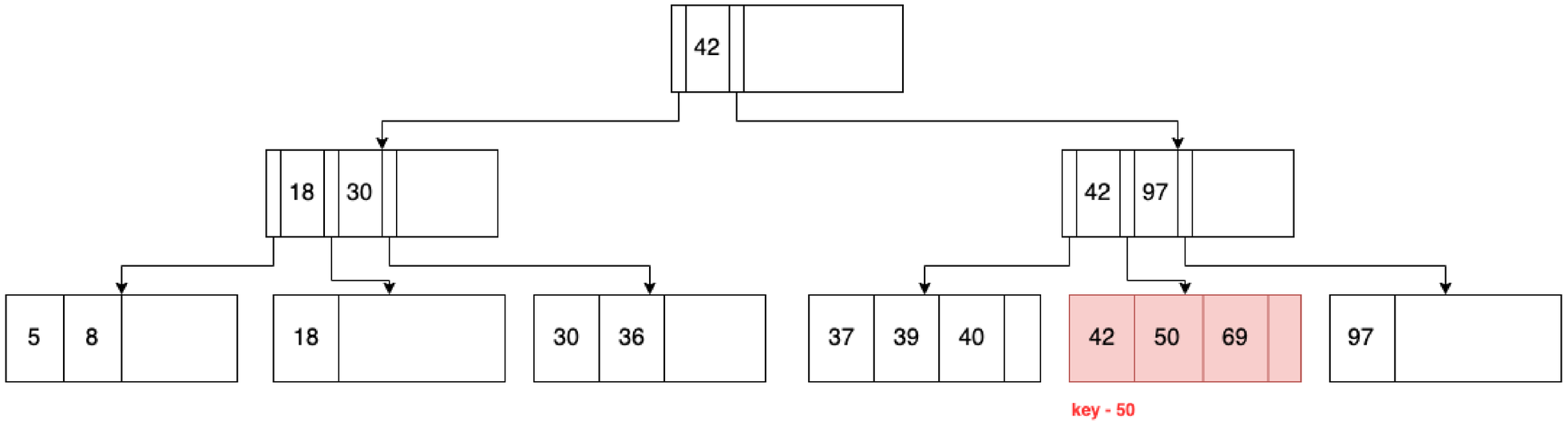

42
key - 50
18
30
42
97
5
8
18
30
36
37
39
40
42
69
97


42
18
30
42
97
5
8
18
30
36
37
39
40
42
69
97
key - 50
contains = false

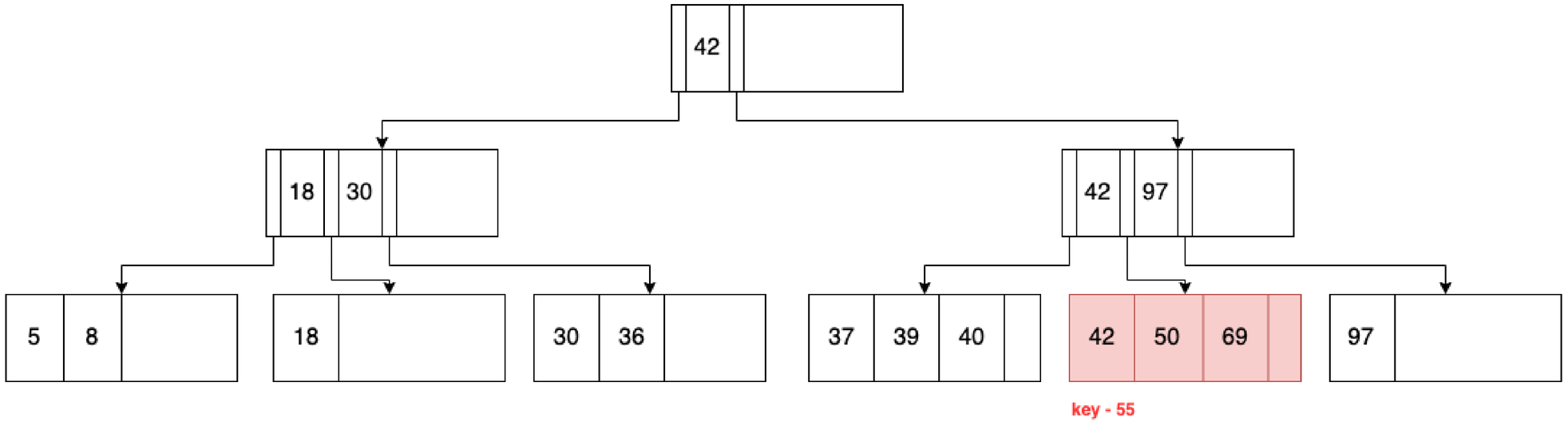
Insertion - happy path

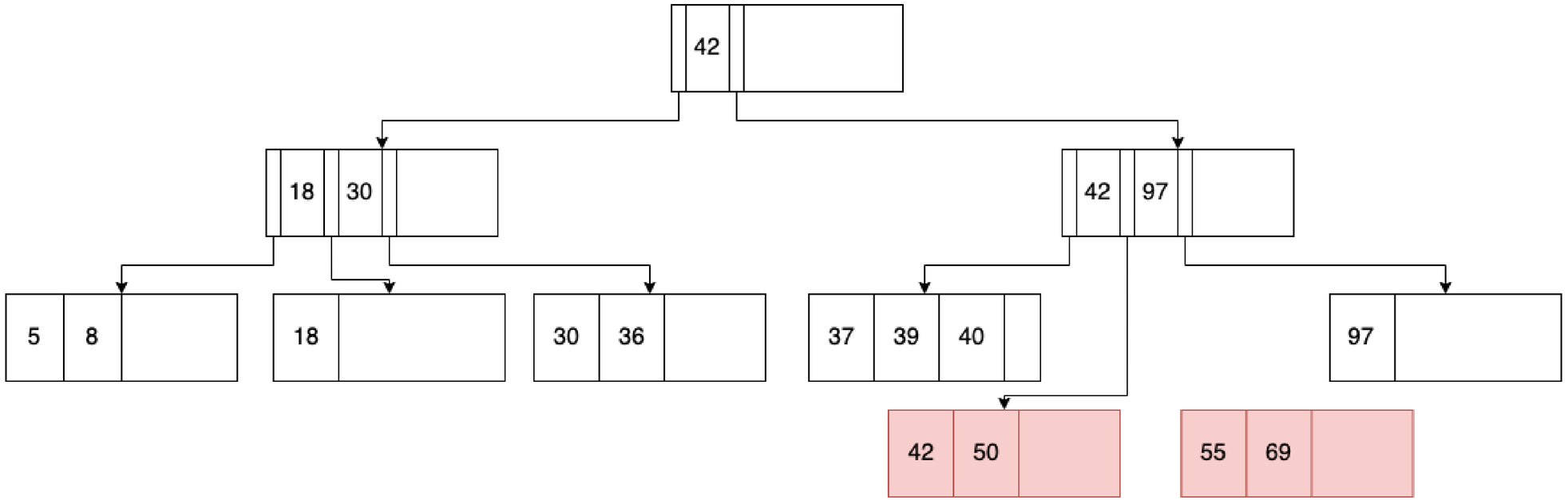

key - 50
42 > 50 ?
42
18
30
42
97
5
8
18
30
36
37
39
40
42
69
97

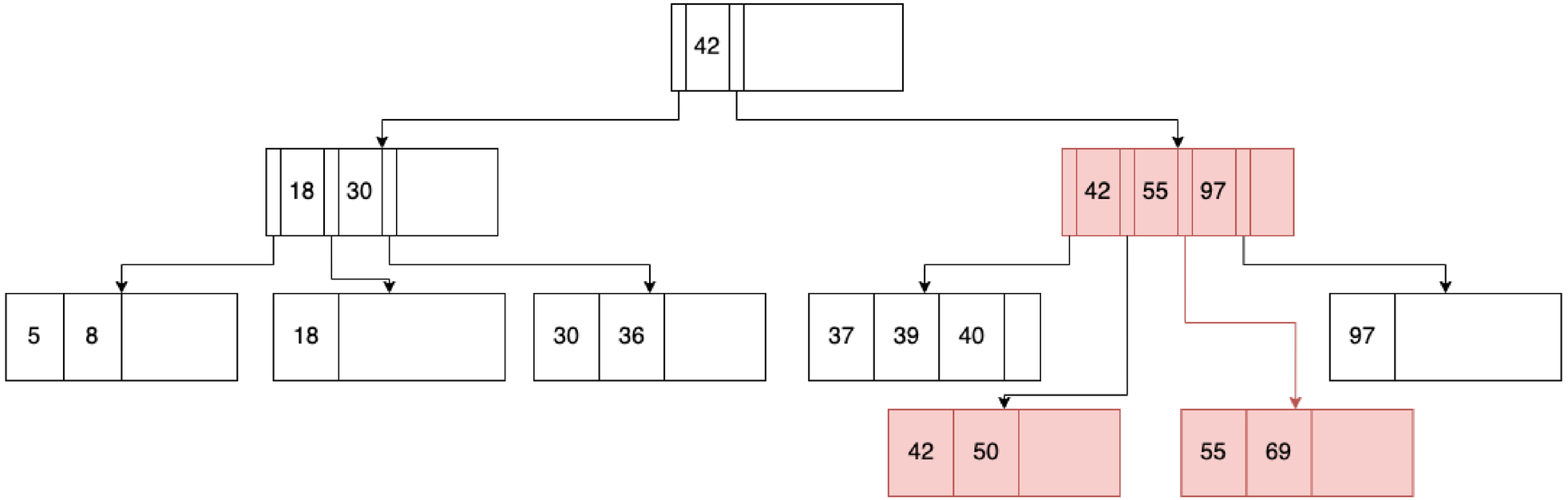

42
key - 50
18
30
42
97
5
8
18
30
36
37
39
40
42
69
97


42
18
30
42
97
5
8
18
30
36
37
39
40
42
50
69
97
key - 50

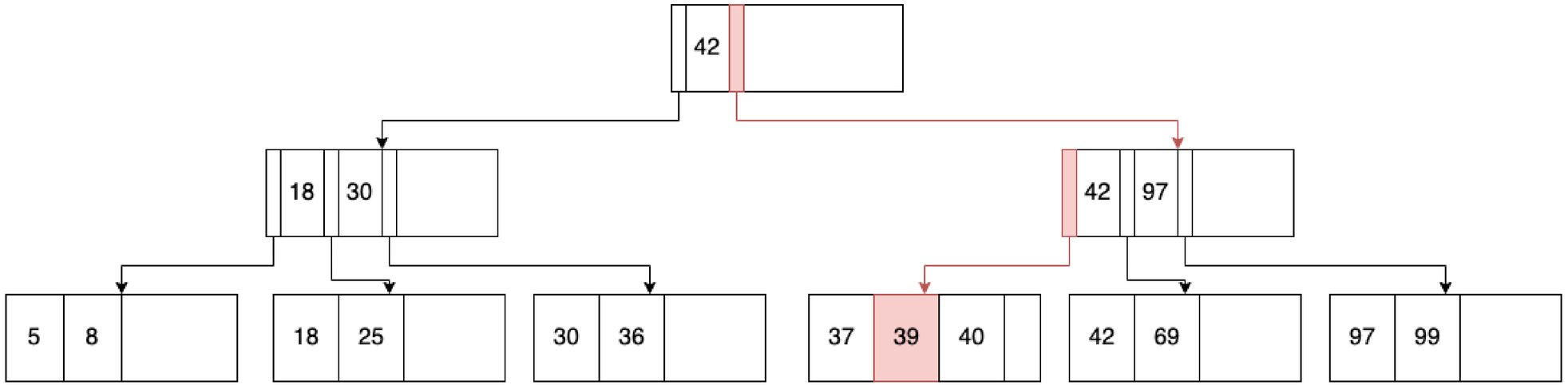
Insertion with split

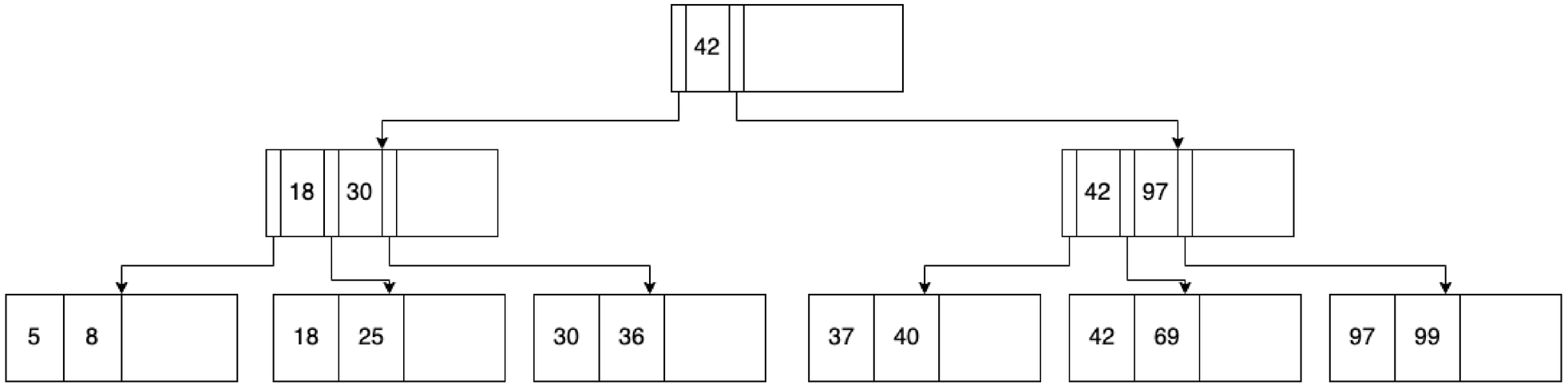

42
18
30
42
97
5
8
18
30
36
37
39
40
42
50
69
97
key - 55


42
18
30
42
97
5
8
18
30
36
37
39
40
97
42
50
55
69

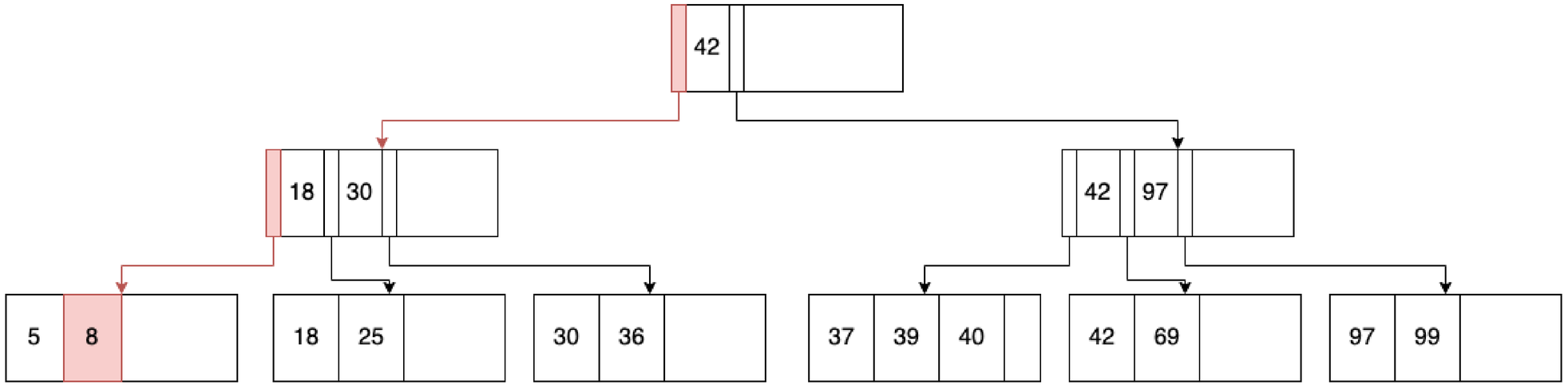

42
18
30
42
55
97
5
8
18
30
36
37
39
40
97
42
50
55
69

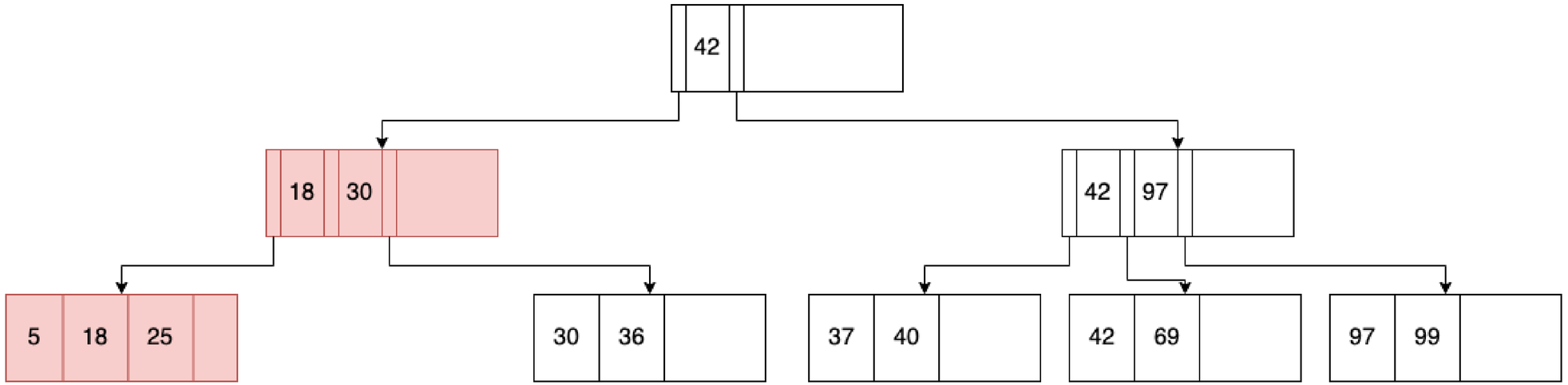
Removal - happy path

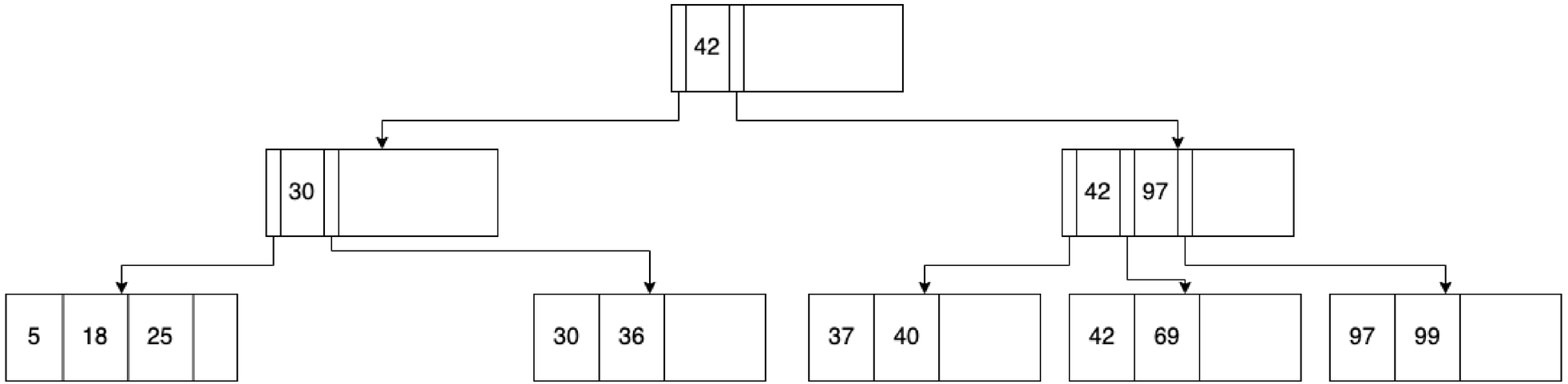

42
18
30
42
97
5
8
18
25
30
36
37
39
40
42
69
97
99


42
18
30
42
97
5
8
18
25
30
36
37
40
42
69
97
99

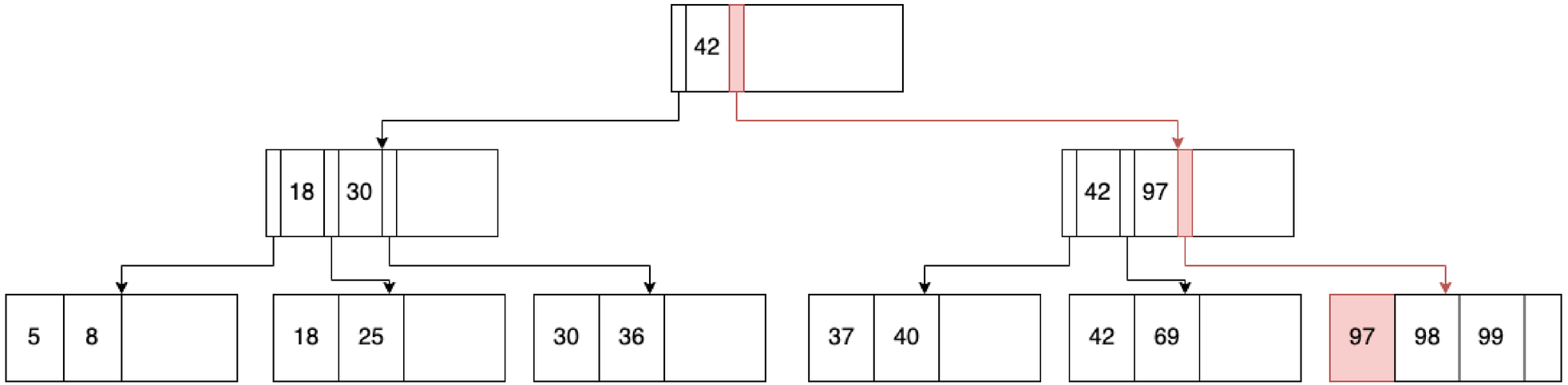
Removal with merge

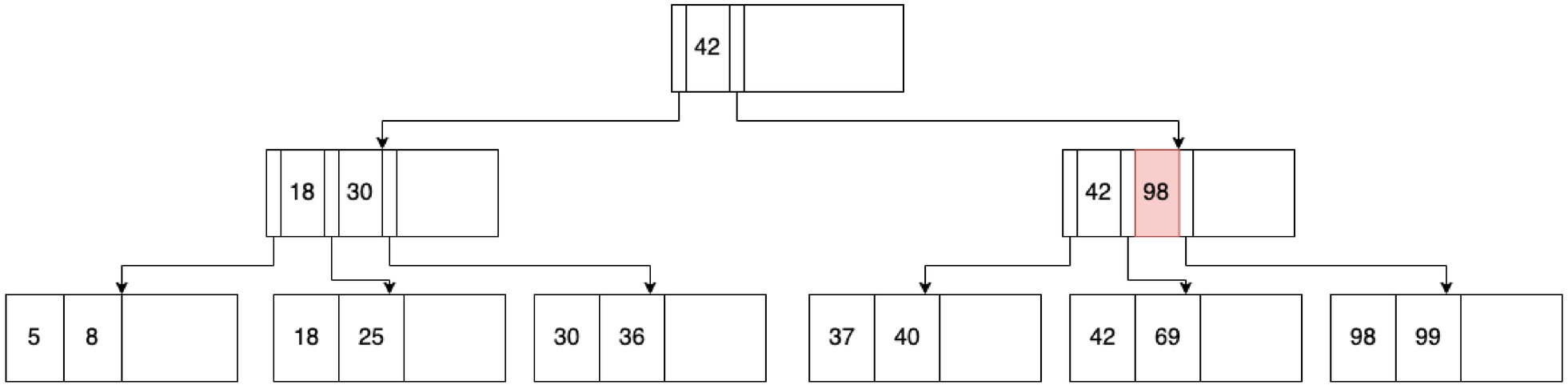

42
18
30
42
97
5
8
18
25
30
36
37
39
40
42
69
97
99


42
18
30
42
97
5
18
25
30
36
37
40
42
69
97
99


42
30
42
97
5
18
25
30
36
37
40
42
69
97
99

Removal - leftmost item


42
18
30
42
97
5
8
18
25
30
36
37
40
42
69
97
98
99


42
18
30
42
98
5
8
18
25
30
36
37
40
42
69
98
99

Searching key inside tree node

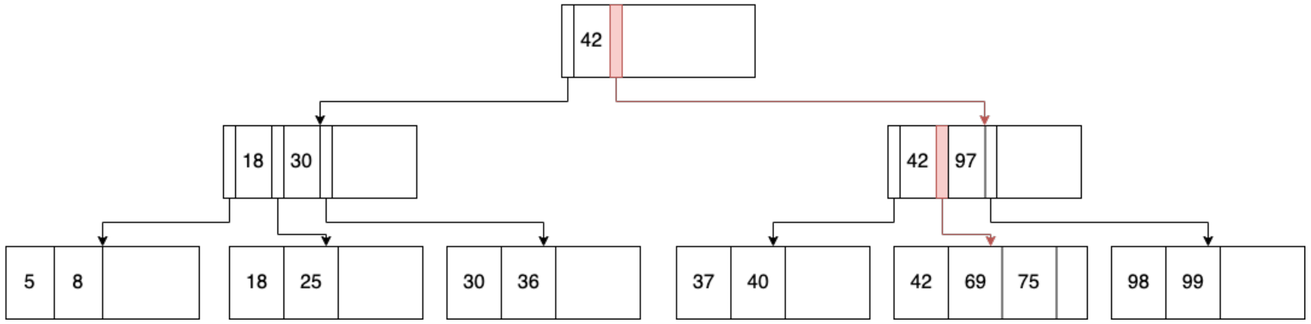
- If the keys has constant sizes use a binary search algorithm.
- Otherwise, just iterate the keys one by one.

Concurrent tree design

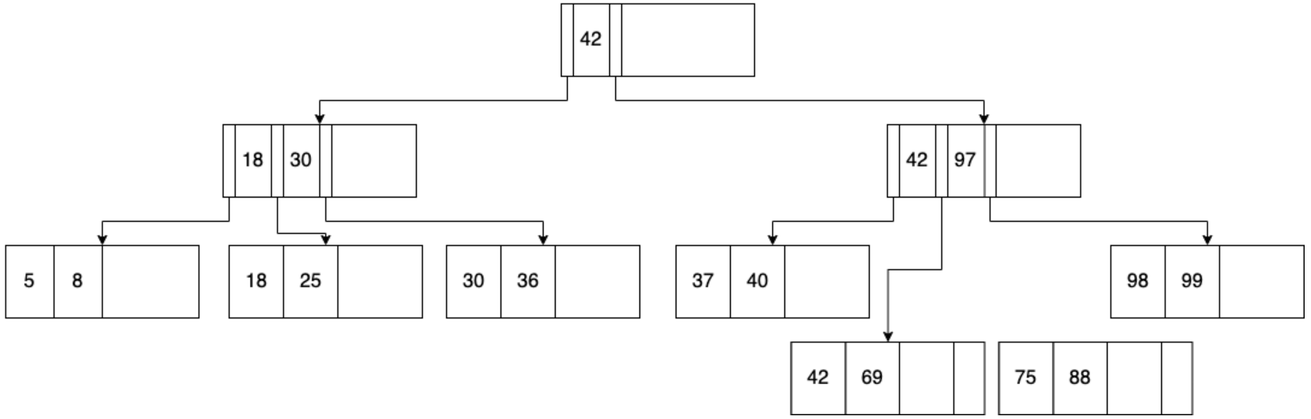
Concurrency issues

- Several threads modify the same page.
- Concurrent operations modify same nodes.
- Basic operations implementations propagate bottom-up.

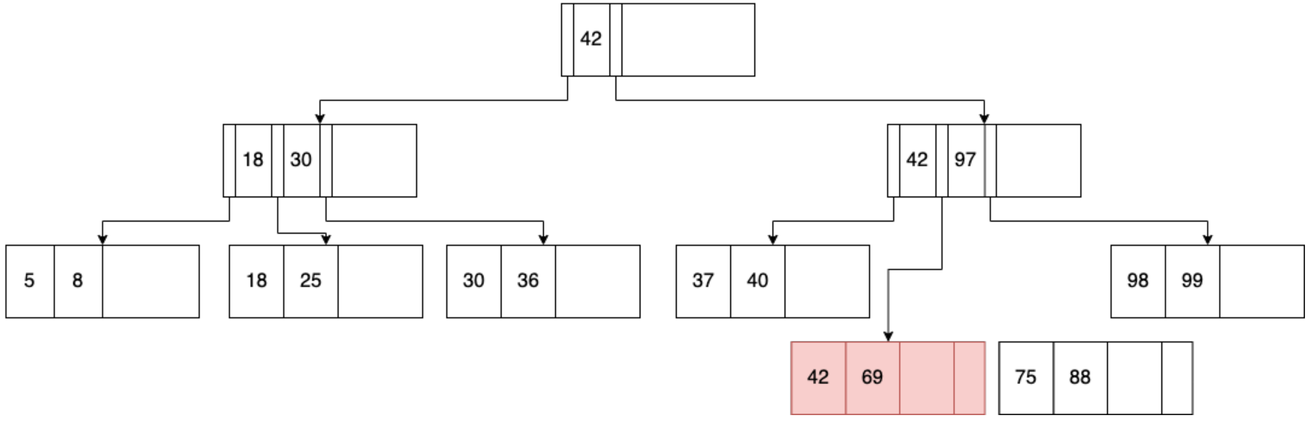
T1 - search 75



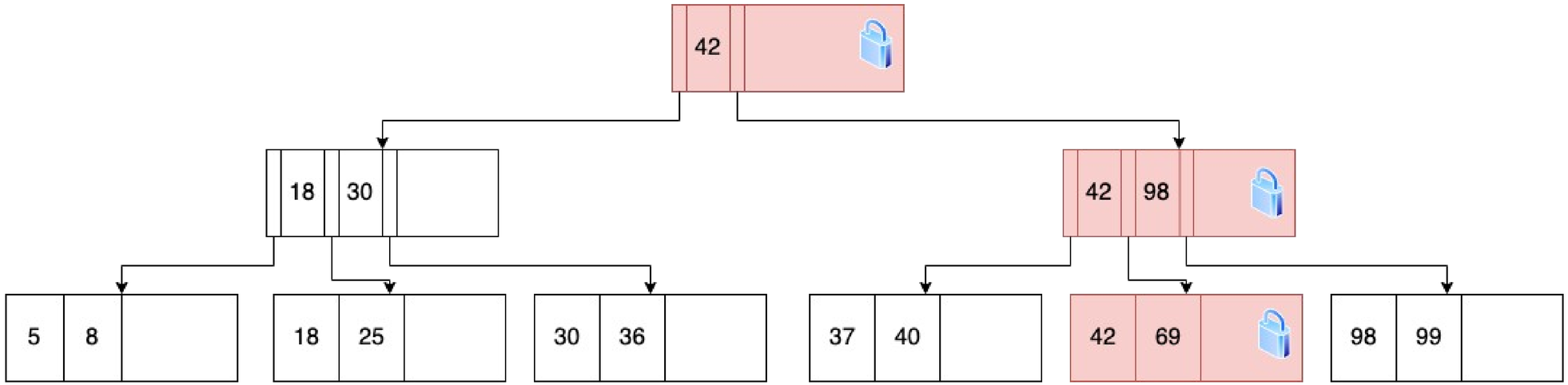
T2 - inserts 88



T1 - reads page.



Naive approach - lock tree path



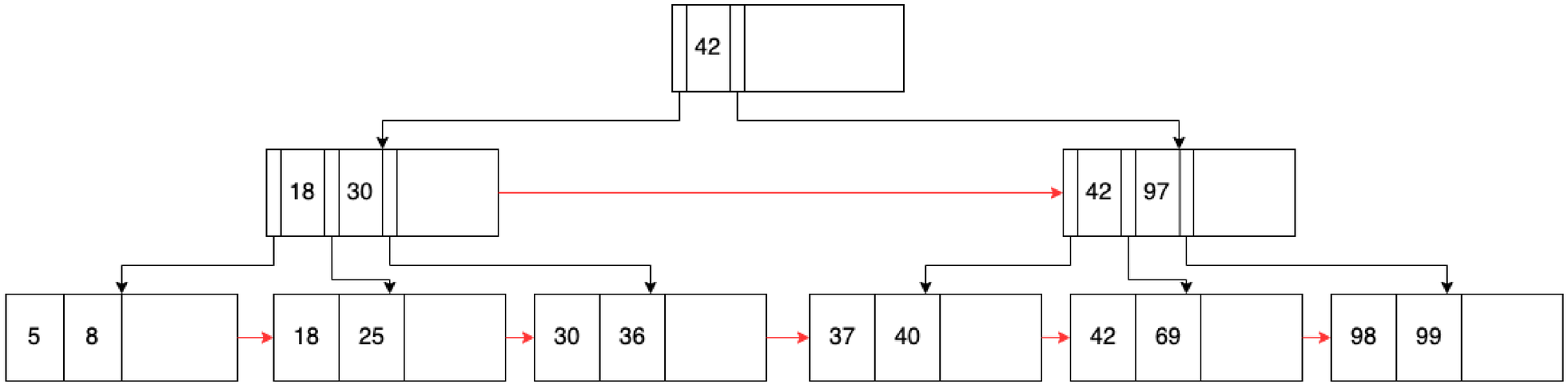
Improved naive approach - optimistic descent

- Read locks during traverse.
- Write lock on the leaf.
- Check for a split/merge.
- If yes, restart with write locking entire subtree.

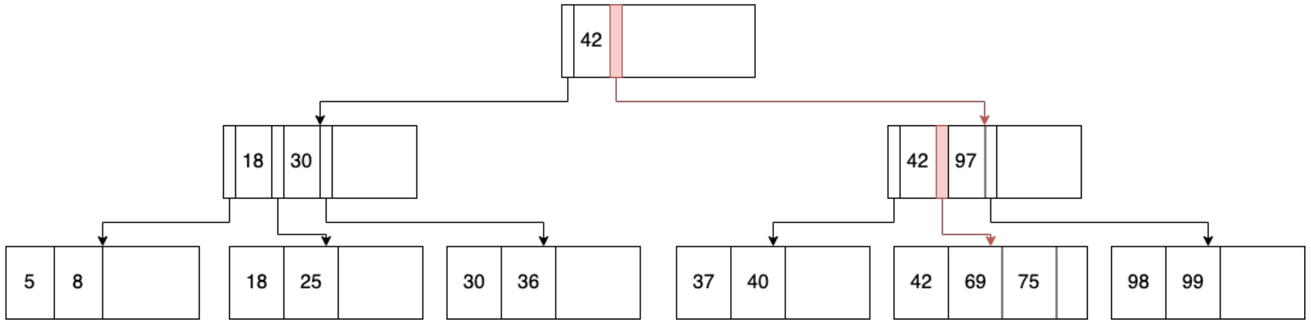
Lock model

- Each page has regular ReadWrite lock.
- Thread gets read lock - guarantee that page will not be modified concurrently.
- Any number of threads can get read lock concurrently.
- Write lock is exclusive.
- When write lock is acquired no other process reads or writes the page.

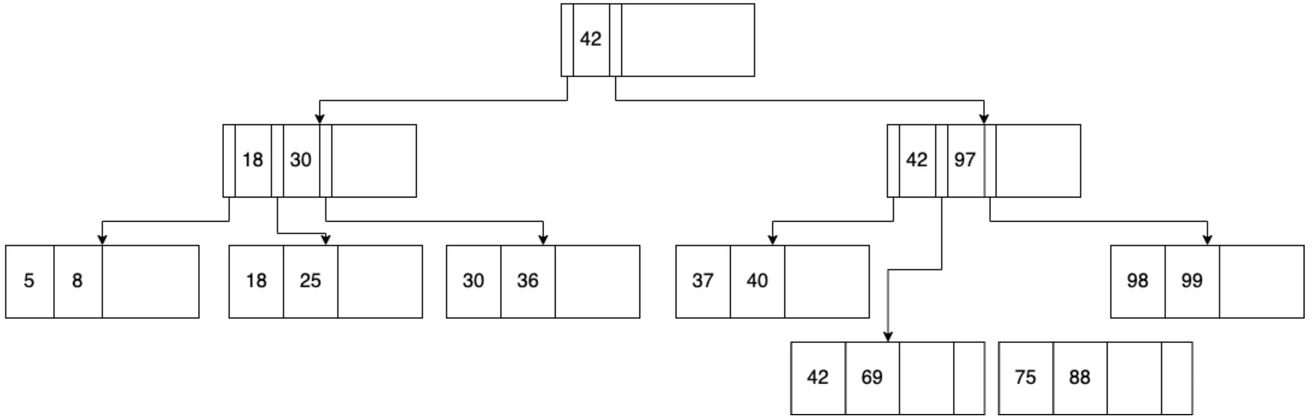
B-link - pointer to the right sibling

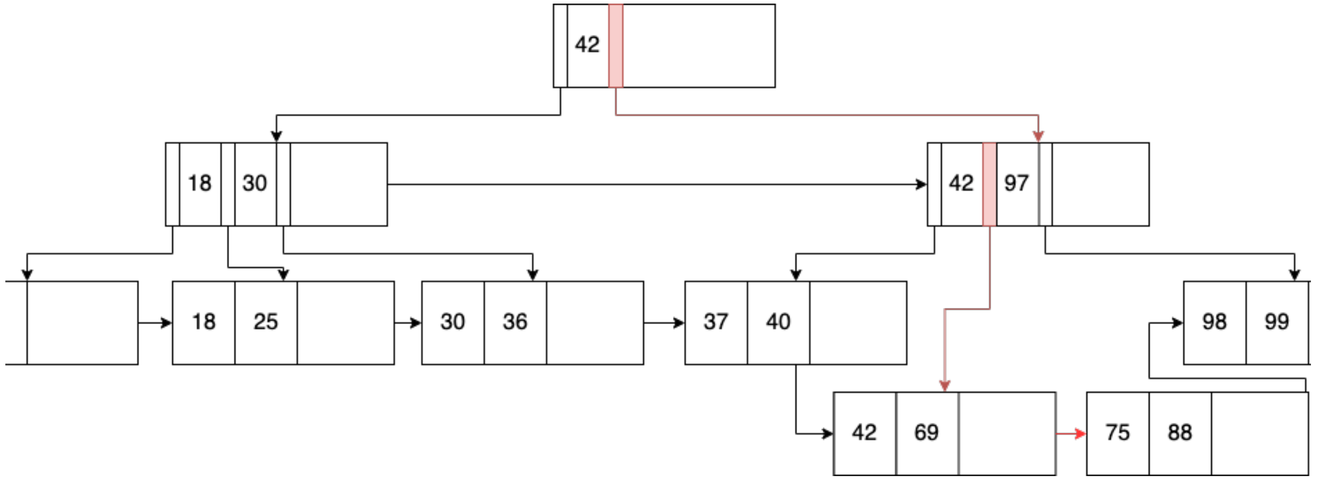


T1 - search 75



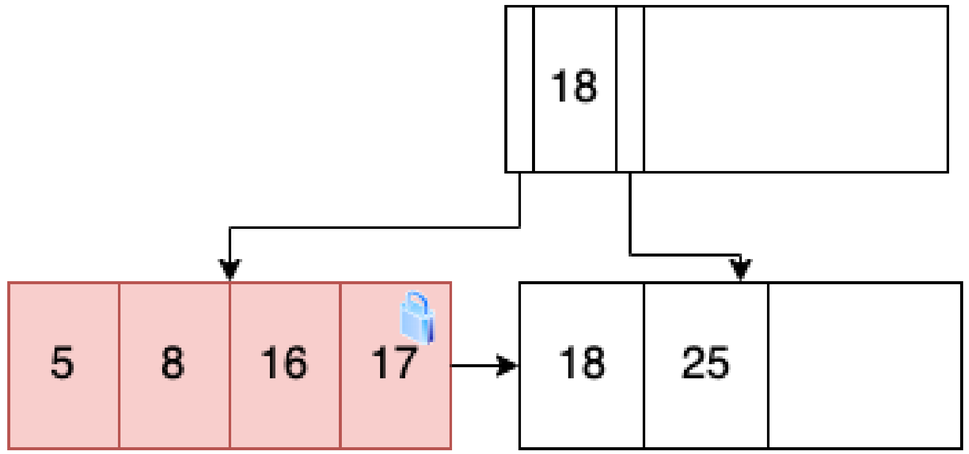
T2 - inserts 88



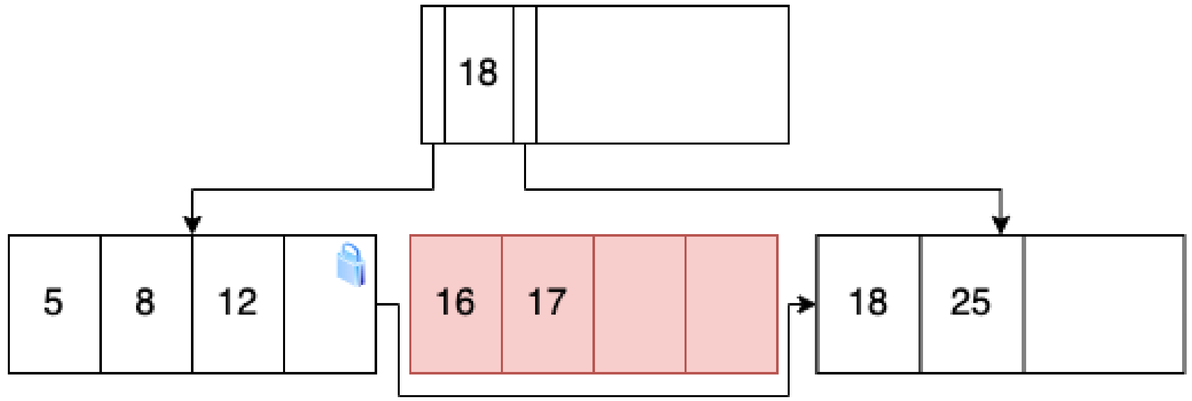


Split algorithm - insert 12

1. Take write lock.

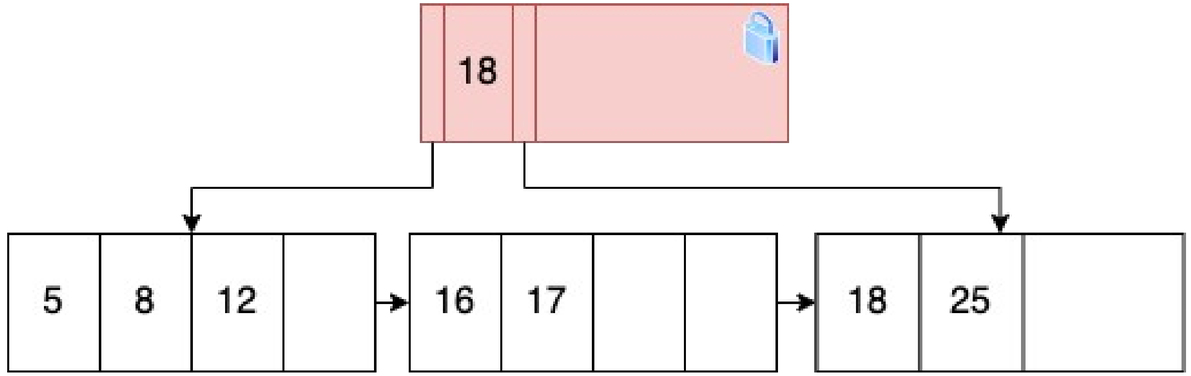


2. Split page.

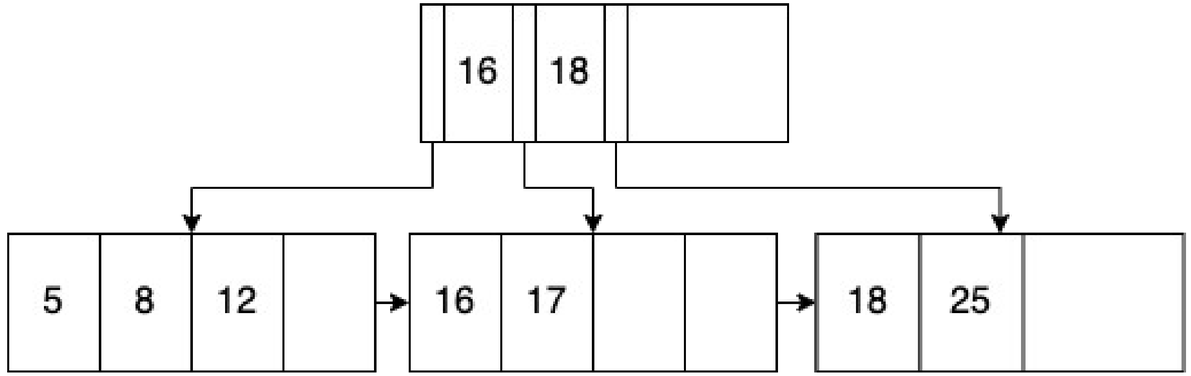


Split algorithm - insert 12

3. Take write lock on parent.

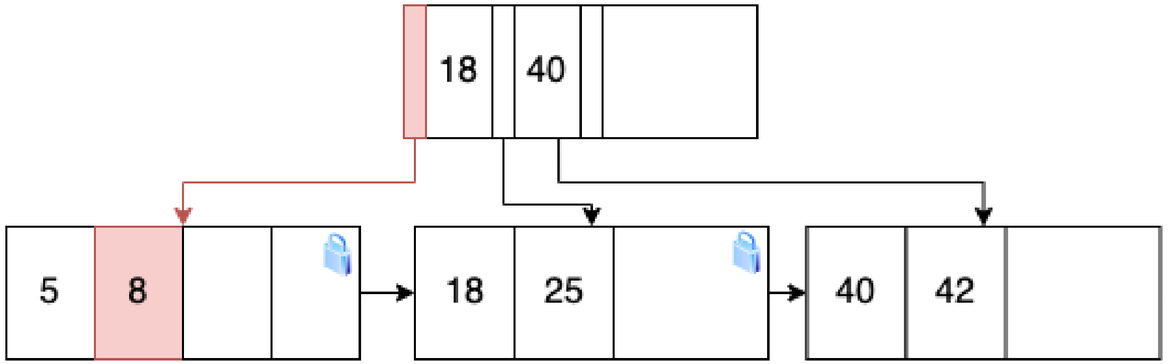


2. Adjust pointers.

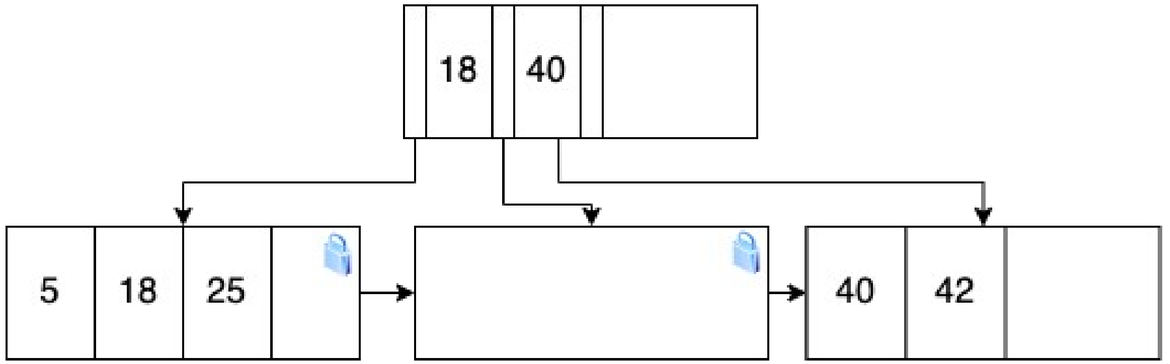


Merge algorithm - remove 8

1. Find an item and take locks.

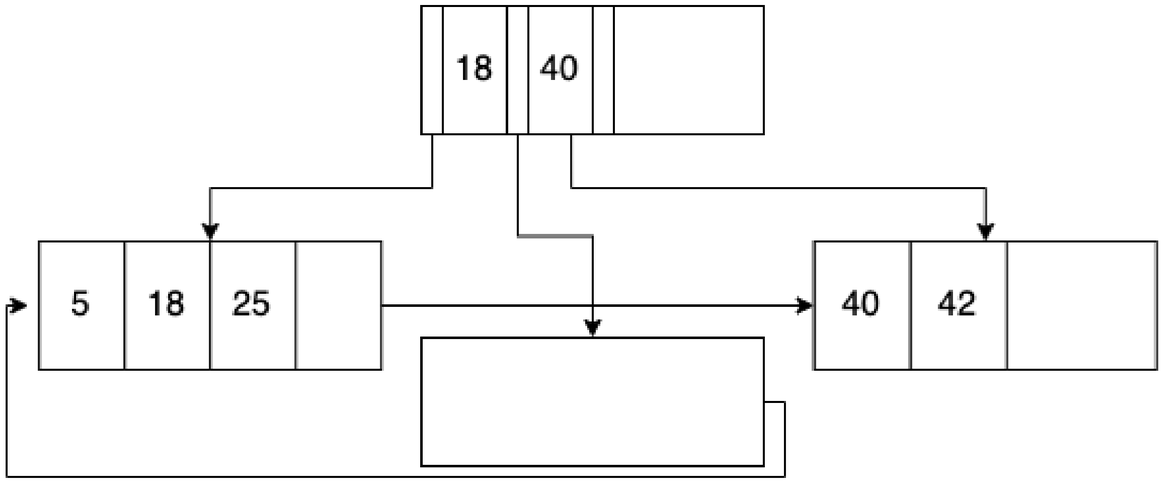


2. Merge pages from right to left.

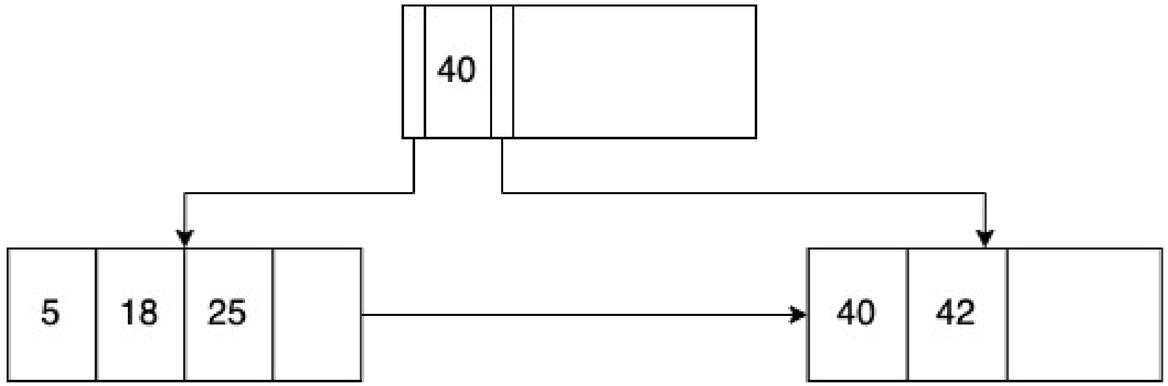


Merge algorithm - remove 8

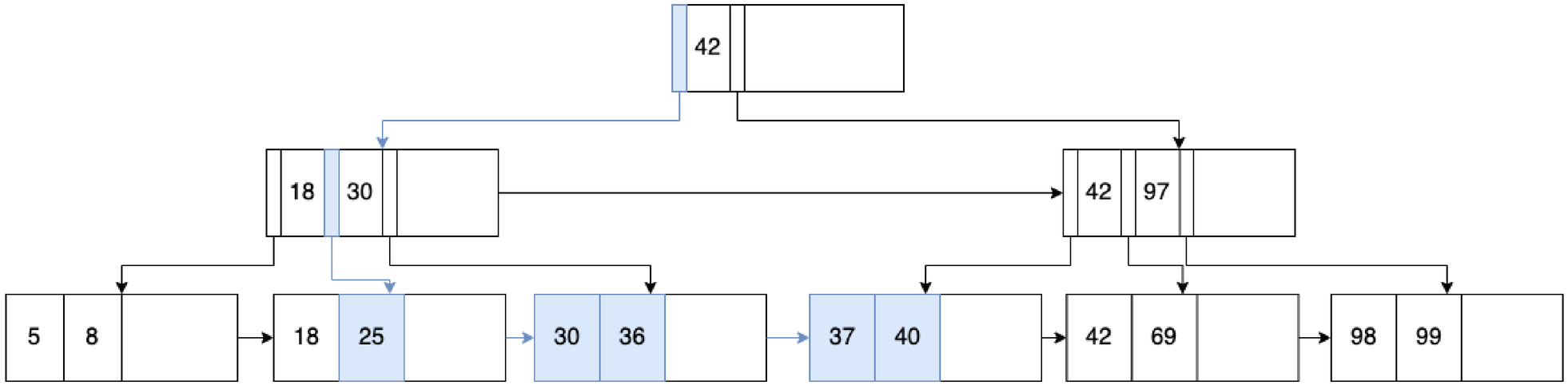
3. Link the empty page to the left precesedor.



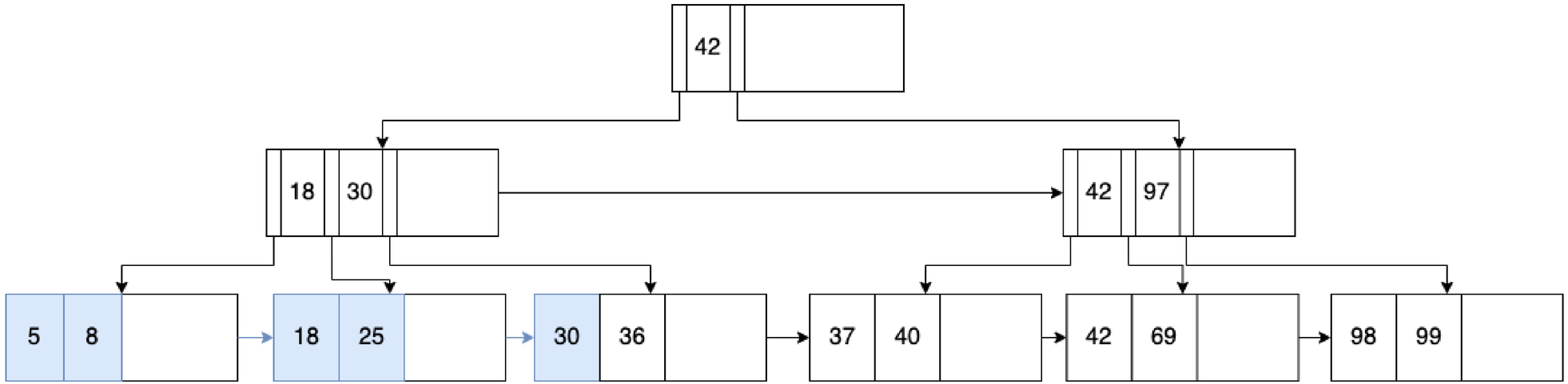
4. Update the parent



Range scan: x > 20 AND x < 40



Range scan: x < 35



Inverted order scan - How to implement ORDER BY x DESC?

- Just make two indexes with inverted order:
  - More space.
  - Easy to implement and maintain.
- Use left sibling link:
  - Less space.
  - Difficulties while scanning and maintaining.

Links

- Efficient locking for Concurrent Operations on B-trees. Lehman, Yao - https://dl.acm.org/doi/pdf/10.1145/319628.319663
- Symmetric concurrent B-tree algorithm. Lanin, Shasha - https://dl.acm.org/doi/pdf/10.5555/324493.324589
- Postgres nbtree docs - https://github.com/postgres/postgres/tree/master/src/backend/access/nbtree
- Ignite BPlusTree implementation - https://github.com/apache/ignite/blob/master/modules/core/src/main/java/org/apache/ignite/internal/processors/cache/persistence/tree/BPlusTree.java

Thank you!